Hidden in plain sight: Camera tracking markers for virtual production setups

Himanshu Pathak

Supervisors: Ruben Wiersma, Elmar Eisemann

Delft University Of Technology

Abstract

With a large move to virtual production in films, where a large screen displaying a rendered scene is used as an alternative to replacing a green screen, there is a need for simple and effective camera tracking. Infrared tracking is widely used in the industry however it creates a barrier to entry for small productions without limited budgets. As an alternative, visual tracking markers (tags) can be used however they come with limitations as they are highly visible and distract the viewer from the film. This paper aims to answer "Can camera tracking fiducial markers (tags) be blended into the background to reduce their visibility whilst still being tracked?". We describe a method to find locations where tags can be placed to reduce their obtrusiveness. Additionally, we explore various blending methods to blend tags whilst retaining their tracking capability. Our findings indicate that blending methods are effective in reducing the visibility of tags yet still be detected but improvements are needed in determining the optimal blending method for different scenarios.

1 Introduction

Films and TV Shows are transitioning from a workflow that required large post-production efforts to a virtual production workflow where massive screens are used to simulate an environment in real-time. This is in contrast to creating replica landscapes in a studio or editing actors in front of green screens [1, 2]. The simulated environment is created using a game engine where a virtual camera renders its view to the screens, which creates the effect of having the actors being virtually present in the simulated world. Designers and directors can make quick changes to the environment, scenery, and lighting whilst seeing the effects live.

One of the difficulties with this approach is tracking the position of the real camera which is used to record the actors. As when this real camera moves the image shown on the screens should move accordingly, otherwise there will be a disconnect from the perspective and the visual appeal is lost [2]. Currently, in the industry infrared sensors are used to detect the movements of the camera which are translated to the movement of the virtual camera used to render the screens [1]. This, however, can be an expensive approach and it would prove beneficial to develop a method to achieve the same effect with limited instruments.

Numerous solutions such as ARTag [3], C^2Tag [4] and ARUco are available for use in camera tracking applications. However, the limitation of these systems in the current state

is that they require visible fiducial markers (tags) to track the camera position. These tags can be visually distracting as they are usually in grayscale with contrasting colors to increase their visibility.

The Mandalorian TV series is one such production that utilizes the Unreal Engine and large LED walls to create visually appealing and realistic scenes [1]. The Mandalorian uses infrared camera tracking as visual camera trackers would be intrusive and require extra effort during post-production. Augmented Reality applications also make use of tracking markers, such applications could also benefit from tags that are hidden.

This research aims to answer the question "Can camera tracking fiducial markers (tags) be blended into the background to reduce their visibility whilst still being tracked?". This is done by exploring whether popular blending modes can be used in conjunction with fiducial markers (tags) to track the real camera through the virtual world whilst not being visually distracting. This paper focuses on the following components.

1. We investigate optimal placement of tags as not all positions may be ideal.
2. Compare numerous blending methods to disguise the tag in the scene.
3. Show that tags can be recovered after being blended.

2 Method

To answer the research question various methods of overlaying the fiducial markers on the background image to be shown on the screens will be explored. The tags used are ARUco tags provided in the OpenCV library. Other fiducial markers solutions utilize tags that are similar to ARUco, made up of a simple predefined shape and contain encoding of data inside with the contrast of black and white pixels. ARUco tags are black squares with binary codes inside denoted by white shapes like the one shown in Figure 1. For simplicity in implementation and due to their widespread use, ARUco tags are used in this research. Additionally, fiducial marker detection algorithms perform optimally when the tags are visible with high contrast to surrounding pixels, techniques will be explored to increase the contrast of tags for more reliable detection. To test the efficacy of different overlaying methods the tags will be placed on a test scene created in Unreal Engine, Figure 2.

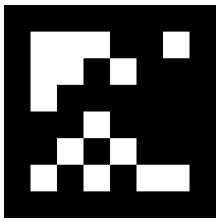


Figure 1: Example of an ARUco tag which will be added to the scene

2.1 Location Selection

The position where a tag is placed is crucial. In some cases, if the position is randomly selected and it lands between large visual transitions in the scene such as the transition between sky and land, the blending methods do not perform well and larger sections of

Figure 2: Test scene created in Unreal Engine emulating a scene with actor and landscape

the marker are lost. Additionally, it is preferred that the tags are closer to the edges of the screen as they are less likely to obscure important parts of the scene and cause visual distraction. The position of the tags is obtained by generating random positions which are bound by two constraints.

The first constraint limits the randomly selected position of the tag to be within the edges of the screen. This is accomplished with a 'margin size' input which can be specified by the user in addition to a 'marker size'. These two parameters are taken into account to bind the random generation within bands around the edges of the screen. The marker size is used to ensure that the tag is positioned within the screen and does not extend outside. One benefit of placing tags along the edges of the screen is that often during post-production in movies the video recorded by the camera are cropped for aesthetic purposes. Occasionally, black bars (letterbox) are added to the top and bottom of the scene to create a cinematic effect. Therefore in many cases, the tags would not be visible in the final movie while still providing the tracking benefit during production.

The second constraint attempts to ensure that the tag is not placed in positions on the screen where there is a large visual transition present, for example in transitions between sky and land or between the sky and water. To find an optimal location a loop is run until a desirable location is found. The method to evaluate the desirability of a location is done by calculating the color difference (ΔE) [5] for a region $markersize * markersize$ anchored at the randomly selected position. For each pixel in the region, the ΔE is calculated to the pixel at $(rand_x, rand_y)$. Subsequently, the ΔE for the whole region is calculated and a region is selected if a standard deviation of 3 or better is achieved. L*a*b* (referred to as "lab" ahead) color-space is used for ΔE calculations in place of taking the color difference in RGB [5]. Lab color-space is preferred to RGB as it encapsulates the entire range of human perception by including lightness (L*), green-red pair (a*), and blue-yellow (b*). Additionally, ΔE values in the lab color-space correspond to perceived color difference to the eye allowing for us to find large transitions in the scene. Once a suitable position is found the tag is blended at that position using blending modes described ahead.

2.2 Blending Modes

In image processing, several methods can be used to overlay one image on top of another. These methods aim to blend the top image onto the bottom image by using the pixel values on the two images. In our case, we try to blend a tag onto the background image to reduce its visibility to the eye whilst trying to keep the source background image as similar to the original as possible.

2.2.1 Multiply and Screen

One of the simpler ways to overlay two images is to take their pixel values and multiply them with each other to obtain the resulting pixel value [6]. This is known as the multiply method. Any values outside of the valid range for images (0-255) must be clamped between this range. In the multiply method, darker parts of the source images result in an even darker pixel value in the resulting image.

On the other hand, the screen method acts as the inverse of multiply. The screen method is described by $f(a, b) = 1 - (1 - a)(1 - b)$ [6], where a and b are the pixel values for the top and bottom image correspondingly. In screen, darker parts of the source images create a brighter region in the resulting image.

2.2.2 Overlay

Finally, a combination of the two, screen and multiply method can be used to create a desirable effect where tags blend into the background image. The overlay method uses the top image a as a mask, wherever pixel values of a are below 0.5, multiply is applied, otherwise screen. In our case a would be the tag and b is the scene.

$$\text{overlay(a,b)} = \begin{cases} 2ab & \text{if } a < 0.5 \\ 1 - 2(1 - a)(1 - b) & \text{otherwise} \end{cases} \tag{1}$$

[6] With the overlay blending the colors of the background are still preserved in addition to the dark and bright parts of the tag.

Figure 3: Tag applied to scene with overlay blending method

2.2.3 Photoshop Soft-Light

The soft-light blending method used in Photoshop is a blending method which shifts both black and white parts of the top image towards the background image [7].

$$\text{photoshop(a,b)} = \begin{cases} 2ab + b^2 \cdot (1 - 2a) & \text{if } a < 0.5 \\ 2b(1 - a) + \sqrt{b} \cdot (2a - 1) & \text{otherwise} \end{cases} \tag{2}$$

[7]

Figure 4: Tag applied to scene with photoshop blending method

2.2.4 W3C Soft-Light

The W3C Soft-Light blending method is similar to the photoshop method however it takes input of both top and bottom layers, using them as masks [8].

$$\text{w3c}_f\text{(a,b)} = \begin{cases} b - (1 - 2a) \cdot b \cdot (1 - a) & \text{if } a \leq 0.5 \\ b + (2a - 1) \cdot (w3c_g(b) - b) & \text{otherwise} \end{cases} \tag{3}$$

[8]

$$\text{w3c}_g\text{(a,b)} = \begin{cases} ((16b - 12) \cdot a + 4) \cdot a & \text{if } b \leq 0.25 \\ \sqrt{a} & \text{otherwise} \end{cases} \tag{4}$$

[8]

The result of the W3C soft-light is a convincing effect where the white parts of the tags brighten the scene and black parts darken the scene. The conditions of the piece-wise functions in both $w3c_f$ and $w3c_g$ can be adjusted in order to control the strength of the blending effect. As can be seen in Figure 5, the dark parts of the scene are blended in the scene whereas the white parts remain only slightly affected.

Figure 5: Tag applied to scene with W3C soft-light blending method

2.2.5 Pegtop Soft-Light

Another blend mode experimented with in this research is the soft-light blend mode used in Pegtop software [7]. This method does not use a mask of pixel values below 0.5 like other method describes, instead it follows a linear function on the top and bottom layer.

$$\text{pegtop(a,b)} = (1-a)\cdot ab + a\cdot(1-(1-a)\cdot(1-b)) \tag{5}$$

[7] Application of the pegtop blending method on the test scene can be seen in Figure 7 in the appendix.

2.2.6 Combination of W3C and Photoshop

As can be seen from the application of previous blend modes, some methods handle brighter parts of tags better than darker parts and vice versa. An ideal tag would be one where the dark and bright parts are both blended into the background. In order to create a more consistent blend mode which achieves this, some combinations of previous methods were created and tested.

One such combination created applies the w3c blending method where $b < 0.5$ and applies the photoshop blending method otherwise.

$$\text{combo(a,b)} = \begin{cases} \text{w3c}_g\text{(a,b)} & \text{if } b \leq 0.5 \\ \text{photoshop(a,b)} & \text{otherwise} \end{cases} \tag{6}$$

This method was tried as in some cases the darker parts of the tag were handled well in the w3c method whereas the brighter parts were more blended into the background in the photoshop method.

The second method is again a combination of *photoshop* and $w3c_f$ where the masking condition and the tag image is altered.

$$\text{combo}_2\text{(a,b)} = \begin{cases} \text{photoshop(a, } \sim\text{b)} & \text{if } b \leq 0.5 \\ \text{w3c}_g\text{(a,b)} & \text{otherwise} \end{cases} \tag{7}$$

where $\sim b$ is the inverse of b. Images of both combo methods are shown in the appendix, figures 8 and 9.

2.3 Post-processing and Tag Finding

Once the tags have been blended into the background scene, tag finding algorithms such as the one implemented in the OpenCV library have trouble finding the tag as there is not sufficient enough contrast within the pixels of the tag. In order to isolate the tag from the other parts of the scene numerous post processing steps are performed onto the tagged image.

1. The contrast of the image is boosted.
2. Square-like objects in the image are detected to aid the tag finding algorithm.

In some edge cases as seen in Appendix 10 OpenCV was not able to locate the ARUco tags, in order to obtain better accuracy square-like objects were first detected and then OpenCV was run in those regions to find the tags. The square objects were found in the scene by utilizing $morphologyEx$ and $find\_contours$ from the OpenCV library. $morphologyEx$ was used to connect components which were imperfect squares such as in cases if a small part of the tag is obstructed and $find\_contours$ located objects which formed closed loops in the scene.

1. The image was converted to gray scale.
2. A threshold was applied to only keep pixels of sufficient brightness
3. A median blur was applied to reduce noise
4. OpenCV $morphologyEx$ is applied to close square regions
5. OpenCV $find\_contours$ is used to locate contours of square objects in scene

Once the contours are obtained they are first filtered to remove contours with a high number of edges as to remove objects which are not candidates for squares. Next, contours are checked to see if their width and height are within 5% of each other. In which case they are considered a square and passed onto OpenCV $aruco.detectMarkers$. Applying the 5 steps described previously showed improvement in the accuracy of finding tags. Images after the application of some steps is shown in the Appendix B.

2.4 Fade over time

A simple step performed in attempt to decrease the visibility of tags even further is fading. Tags are slowly faded in over time at a fixed position on the source scene. After the tag is fully faded in, it is then faded out. When a tag is blended onto the scene instantaneously it can be noticed easily by the viewer even in their peripheral vision. However, when blended slowly over time it can be overlooked if the viewer is focused on action in another part of the scene. Multiple tags being blended in and out at different times may be necessary as detection will not be reliable on tags which are not completely visible.

3 Experimental Setup and Results

In order to test the effectiveness of different blending modes, a test environment must be set up which imitates conditions present during film production. For this research a test scene was created in Unreal Engine depicting a desert environment which was used as the background for blending tags (Figure 2).

One important metric is the difference that is introduced when a tag is added to a scene. This was measured by calculating the ΔE (color difference of pixels) of the scene after the tag was added compared to the original scene. As the ΔE is a value that is minimized during the Location Selection phase, the difference in the ΔE must be measured while the position of the tag in the scene is fixed. For the purpose of understanding how tag sizes affect color difference in addition to blending methods, tag sizes were also varied and tested. Lastly, an average over 1000 repeat experiments are taken to understand the variability and to reduce bias.

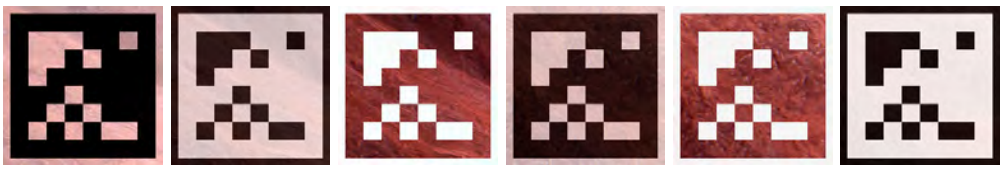
Marker Size (px)	Method					
	Overlay	Photoshop	w3c	combo	combo2	Pegtop
100	32.13	32.65	35.69	29.22	35.45	36.81
150	31.99	32.62	32.33	29.39	32.97	36.96
200	31.43	30.31	28.94	25.98	29.29	35.64
250	29.56	27.80	21.62	22.39	19.92	35.76

Table 1: Average ΔE values over 1000 runs of each method. Repeated for marker sizes in increment of 50 pixels.

From the table 1 we see a lot of variance in the average ΔE values even after 1000 runs. No general trend across different methods can be observed when compared to different marker sizes, this is probably due to the location selection as in different runs, different positions are selected and a larger sample size is possibly required. However, in general as the marker size is increased there is a trend to a lower ΔE. On the other hand, in methods such as $w3c$ and $combo2$ there is a large decrease in ΔE as the $markersize$ is increased, these changes seemed to be outliers however similar results were obtained in repeat runs. In a large image of 3560x2160 pixels there are more than 1000 possible positions where a tag can be placed, however not all positions can be checked due to the time that is taken to place a tag in every position and check its ΔE, therefore a random selection is used to explore the area uniformly. The results show that the w3c, and the two combo methods perform best with tags of larger size and are good candidates for tracking applications. The size of the tags used should be relative to the size of the background image as a large tag on smaller image will cover a larger area which is not ideal.

A lot more can be learned from inspecting the application of the blending modes visually up close as seen in Figure 6. The overlay method in Figure 6a blends the black pixels in the tag into the background whilst keeping detail in those parts however seems to invert the white parts and losing detail. Next, the photoshop method blends both the white and black part of the tag into the background however similarly to the overlay method inverts the whole image. This causes the large black parts of the tag to be shown as white and creating a large visual difference. Combo method performs similar to the photoshop method however without the inverse of the tag, causing a more pleasant blend with the background

in addition to retaining detail in the background. The w3c, and combo2 methods visually perform well and with different characteristics to the others. In these, the parts where the tags are placed seem to accentuate the contrast of the background even more. This can be seen in the white parts of the tag in Figure 6c and Figure 6e, where the background is clearly visible with higher contrast. Lastly, the pegtop method applies a blending similar to the photoshop method with a less pronounce blend. In this, very little detail is present which is barely visible and seems as if the tag is placed on the image without blending.



(a) Overlay (b) Photoshop (c) W3c (d) Combo (e) Combo2 (f) Pegtop

Figure 6: Close-up of tags blended onto test image.

Testing the tag finding was performed by storing the positions tags were added as ground truth which were then compared to the positions reported by the tag finding algorithm. Similar to the test of color difference, marker size was varied in conjunction with the blending method to understand the effect of both parameters. In cases when markers were too small such as 50 pixels in a 3840x2160 pixel, the marker was not found by the tag finding algorithm. Data presented in Table 2 shows the number of images where tags were correctly found over 100 runs. In all blending methods most tags are found except for the overlay method where a lower number of tags were found. It is important to mention that the number of square-like objects (objects with their width and height within 5% of each other) detected in each image is higher than 1. From the data, it may be concluded that the tag finding algorithm performs sufficiently well under all conditions given that the tag is of sufficient size however in more complex scenes perfect accuracy may not be achieved, further testing is required. In addition to this, all testing was performed with the image perfectly aligned with the camera, however, in practice this is not the case. When the camera is off-axis the image along with the tag will be skewed, in those cases, the algorithm will not detect tags if they are largely skewed as tags which are not square-like are rejected. For future improvements, when a contour is detected by OpenCV with 4 sides it can be projected to a square shape in order to detect tags in cases when tags are recorded off-axis.

Marker Size	Method					
(px)	Overlay	Photoshop	w3c	combo	combo2	Pegtop
50	70	100	99	99	99	100
100	98	100	100	99	99	100
150	96	100	100	95	100	100
200	99	100	100	99	100	100

Table 2: Number of times tags were found compared to ground truth in image with 1 tag placed over 100 runs of each method. Repeated for marker sizes in increment of 50 pixels.

In order for the methods described previously to be used in a virtual production workflow, the methods must be performed with speed. For each frame of input, the method must find

a suitable location to place the tag, apply a blending method and place the tag and lastly locate the tag. All these steps must be performed at 24 frames per second or within 41.6 milliseconds as most films are recorded at this frame rate. The code was implemented in Python and was run 7 times with 100 loops in each run. The reported time taken is 53.7 ms ± 3.41 ms per loop (mean ± std. dev. of 7 runs, 100 loops each) on a machine with a i7 6-core processor. Therefore on this machine, the method cannot be run in real-time applications however as the methods were implemented in Python using Numpy and OpenCV, the program can be ported to faster languages such as C++ or C, which may reduce overhead. Additionally, OpenCV can be greatly accelerated with the use of a graphics card therefore real-time speed can be achieved with powerful hardware.

4 Responsible Research

It is important to conduct research whilst keeping in mind the implications of the research published, as inconsistencies or miscommunication in any sections of the paper can result in issues. For this reason in this research paper the methods conducted and tuning parameters used to obtain the results are described in detail. This is done to ensure the reproducibility of the results and to allow for testing by third parties. In addition to this, all results provided in previous sections are from experiments conducted and data is not left out or picked. However, due to the random nature of some aspects of the algorithm some data if collected independently may vary. The test scene used is created to be a representative scenario of real-world applications in order to understand the methods better. However in practice, film scenes can become very complex and varied with vast landscapes, multiple actors, overwhelming range in lighting, and hundreds of props among other elements, where the algorithm may perform differently than described in the results.

5 Conclusions and Future Work

This research aimed to find whether it is possible to blend fiducial marking tags into images to decrease their visibility whilst not losing their tracking functionality. First, a suitable location was chosen for placing tags by picking a location on the edges of the screen in addition to the color difference ΔE in that region. Subsequently, various blending modes were tested to see whether they can blend the tags into the background scene without losing detail in the scene or the tag. Lastly, tags were recovered by locating square-like objects which were then identified using the OpenCV library. The methods described for placing and finding tags could be useful however the effectiveness of blending modes tested is erratic however promising. If the methods described in the paper are to be used in film production, testing must be conducted prior to find the optimal blending methods and marker size for the scene being filmed. For future research, more blending modes should be tested in addition to implementation GPU acceleration in order to achieve real-time performance. The detection of tags is also an area for improvement as in practice tags are not perfect squares and may be skewed.

References

[1] “Why ’the mandalorian’ uses virtual sets over green screen | movies insider,” Jun 2020.

[2] J. Holben, "The mandalorian: This is the way," *The Mandalorian: This Is the Way - The American Society of Cinematographers*, Feb 2020.

[3] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 590–596 vol. 2, 2005.

[4] L. Calvet, P. Gurdjos, and V. Charvillat, "Camera tracking using concentric circle markers: Paradigms and algorithms," in *2012 19th IEEE International Conference on Image Processing*, pp. 1361–1364, 2012.

[5] G. C. O. T. Collective, W. Collins, A. Hass, K. Jeffery, A. Martin, R. Medeiros, and S. Tomljanovic. BCcampus, 2015.

[6] "Blending modes of photoshop & co.."

[7] J. Gruschel, "Pegtop blend modes," Mar 2006.

[8] "Compositing and blending level 1. w3c candidate recommendation," Jan 2015.

A Images of Blend Modes

Figure 7: Tag applied to scene with pegtop blending method

Figure 8: Tag applied to scene with combo blending method

Figure 9: Tag applied to scene with combo2 blending method

B Images of Tag Finding

Figure 10: Case where OpenCV ARUco failed to find tags using 'aruco.detectMarkers' using default parameters.

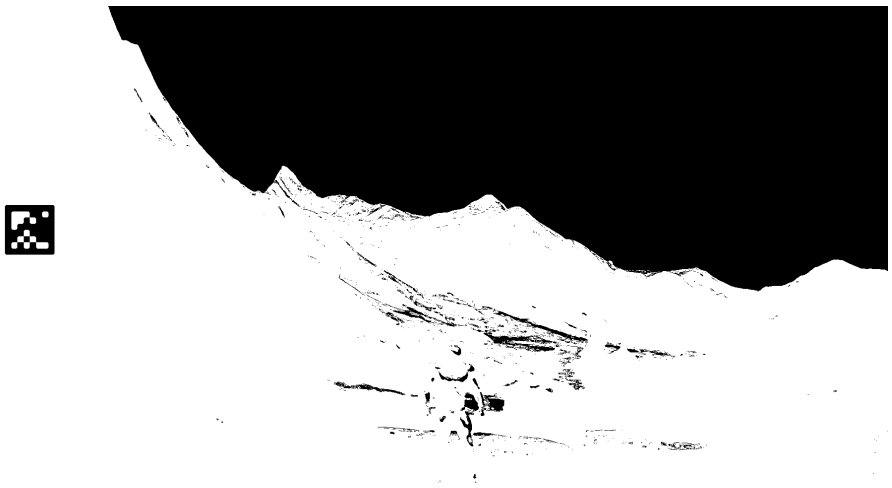


Figure 11: Image after converting to grayscale and applying thresholding.

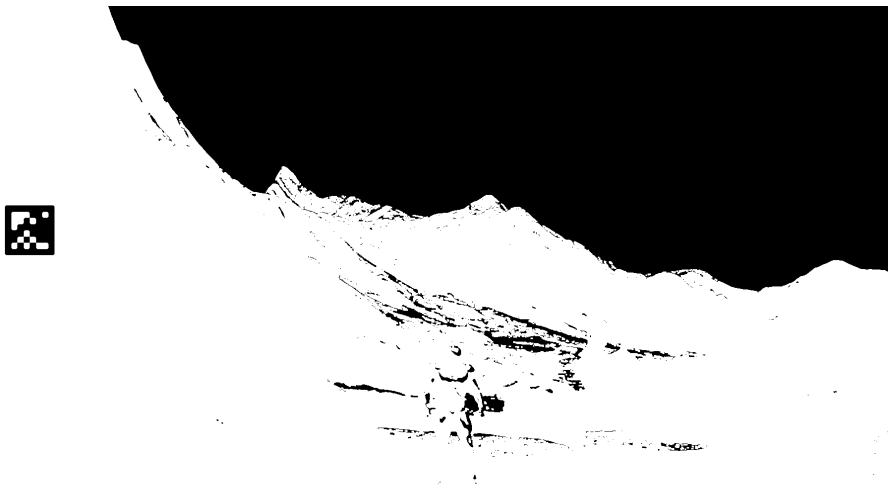


Figure 12: Image after application of 'morphologyEx' using a square kernel.